Self Contained Encrypted Telnet

Nick Christenson, npc@jpl.nasa.gov
Telos Systems Group/Jet Propulsion Laboratory
Pasadena, CA, USA

Abstract

The self contained encrypted telnet, ETN, presented here is designed to provide a secure method of communication between two hosts talking over a potentially insecure network. This package uses the Diffie Hellman key exchange algorithm[1] to negotiate a schedule of 1DES session keys, and uses these to encrypt the entire data stream between client and server. This system is simple to implement, requires no ongoing maintenance and is transparent to the user.

Introduction

Over the past few years, the passive network sniffer has become an increasingly popular method of attacking legitimate computer accounts[2]. It has become clear that more sophisticated solutions are necessary to protect information as it crosses potentially hostile networks. One solution is the use of one-time password schemes, like S/Key[3]. Another solution is the increasing use of encryption in protocols like Kerberos[4]. Yet another solution is to use a secure method of passing the password between the client and server and then run a regular telnet session from then on, as is done in SRA telnet[5]. A one time password scheme solves the problem of eavesdroppers being able to masquerade as a legitimate user and log on to a system, but there are several problems they do not address. For one thing, an eavesdropper can still gain valuable information about the connection. The username is still visible. The natures of other connections are still apparent. The contents of the user's session are not protected. Key distribution based encryption services, like that of Kerberos, also do a fine job of protecting users' sessions. They have their deficiencies as well, however. There is overhead to maintaining the services. There is a heavy reliance on the robustness and integrity of the key server. Inter-realm communication still requires some extra work. SRA telnet works well, but requires Secure RPC, doesn't protect passwords required by commands like su(1), doesn't protect you if you connect over multiple hops (like if you use a gateway) and uses Diffie Hellman keys that may not be sufficiently secure. These schemes solve the problems they address, but I believe there is a niche open for another solution.

Goals

The goals of ETN are as follows:

1. Be easy to build and install.
2. After installation, require zero maintenance time.
3. Have the operation be completely transparent to the user.
4. Require minimal startup time and provide acceptable throughput, even on slow hardware.
5. Provide complete protection against eavesdroppers attempting to discover username/password pairs or any other data communicated over the network during the connection.

Protocol

Here's how it works. The client, is given the name of the machine to which the user wishes to connect. The client requests a particular session key negotiation algorithm from the server, which may concur or demand the adoption of a different algorithm. Currently, 512 bit Diffie Hellman with a client transmitted prime and generator, called `DHA0512` in the ETN protocol, is the only one included, but it would be straightforward to add others. In `DHA0512`, the client starts by sending the Diffie Hellman prime and generator to the server. Then the client and server each randomly generate public and private keys. After they exchange public values, each computes the agreed key based on the information they know: their private value, the other member's public value, the session prime and the session generator. The agreed key is transformed into a schedule of DES session keys to encrypt and decrypt the entire session (Figure 1).

Note that at no time is any additional input required of the user other than what would be required for a regular telnet session. Additionally, there are no extra requirements on the system such as all appeal to a trusted key server or action on the part of some third party. Finally, the performance is good. Even slower workstations can provide excellent throughput. While the public key encryption algorithms slow down the initial connection, no big primes need to be generated in real time, and the key exchange is only donw once.

Requirements

The client and server are based on the Kerberos ready **4.3** BSD Net2 `telnet` and `telnetd` source code. Sites wishing to build ETN also need the RSAREF 2.0 library, which contains the Diffie Hellman code, and the Cygnus Kerberos v.4 `libdes.a` library, which contains the DES code. Instructions on how to obtain these libraries are available in the ETN source package. Although not built yet, it is likely that the EFTP (Encrypted FTP) client will be built from Berkeley source and the server will be adapted from the DIKU `ftpd` code.

Vulnerabilities

ETN does not solve every problem, nor is it designed to. Unlike Kerberos, ETN does not guarantee that the remote host you are talking to is the one you intended. If someone is able to masquerade as your destination host, they may be able to convince you to disclose the data you are trying to protect. Also, if the intruder is able to intercept your data line before the packets reach their intended destination, they may proceed with a "man-in-the-middle" attack, where they accept, a connection from you as your destination and establish a connection with your true destination. This intruder can negotiate the protocol with you, receive your decrypted information, turn around and pass that information on to the intended destination and return the destination's data back to you (Figure 2). If the network latency is bad enough and the intruder has a fast enough machine, you might not be able to notice that this is happening at all.

ETN Authentication Protocol

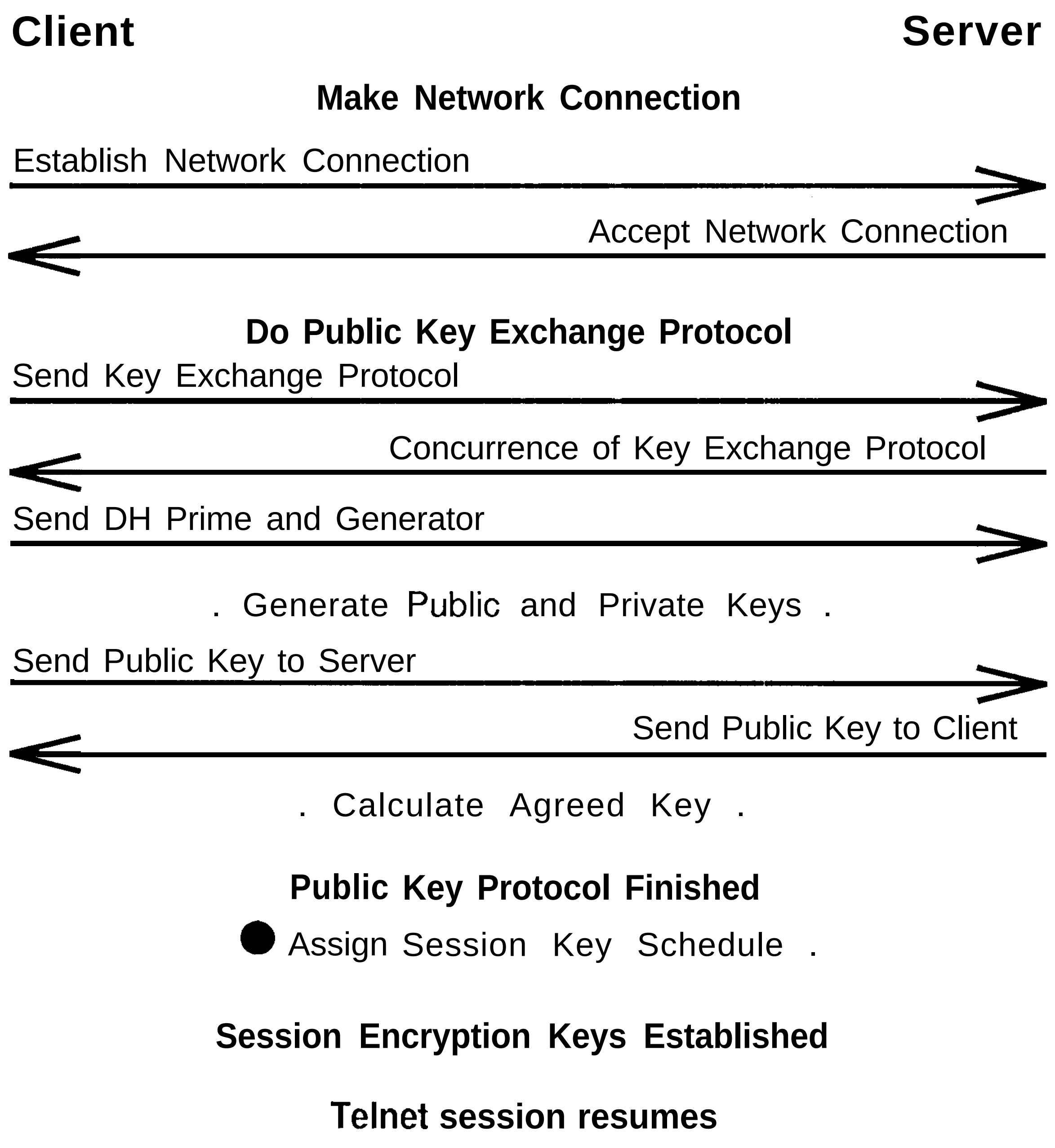
Client
Server
Make Network Connection
Establish Network Connection
Accept Network Connection
Do Public Key Exchange Protocol
Send Key Exchange Protocol
Concurrence of Key Exchange Protocol
Send DH Prime and Generator
. Generate Public and Private Keys .
Send Public Key to Server
Send Public Key to Client
. Calculate Agreed Key .
Public Key Protocol Finished
● Assign Session Key Schedule .
Session Encryption Keys Established
Telnet session resumes

Figure 1

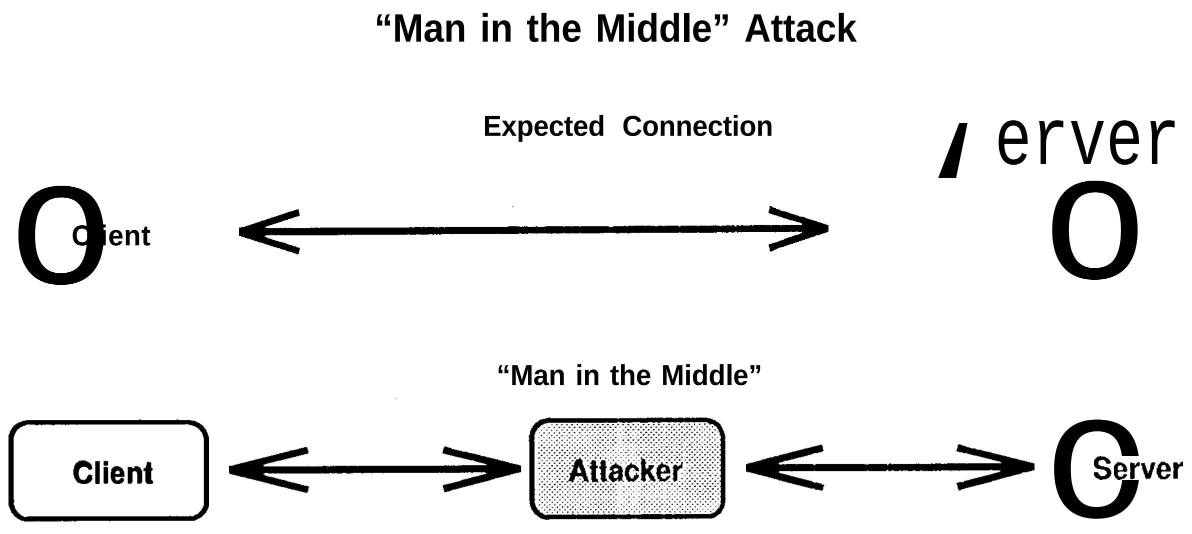

Figure 2

Of course ETN does nothing to attempt to hide traffic analysis information by an eavesdropper, where knowledge that a connection has been made, the 11)$ of the two machines in communication or the time and duration of the connection is the information that is sought. Nor does it' provide any real defense against anyone with permission to read the ETN user's share of machine memory where they could see the plaintext before it becomes encrypted or after it is decrypted.

The problem ETN was designed to solve is the problem of] preventing password network monitoring from gathering information about about' the contents of a remote interactive session. This it does well. If there are stronger requirements, then some other protocol is necessary.

DES and 512 bit Diffie Hellman are considered to be strong but not unbreakable cryptographic algorithms. For example, in 1993 it was estimated that one could build a machine that could exhaustively search through the entire DES key space in 21 minutes for about $10 million using specially built hardware[6]. In software, at best, it would take hundreds of high powered workstations months of dedicated cracking to exhaustively brink I) Es.

It is believed that the time required to factor the discrete logarithms generated from primes of given length, as in Diffie Hellman, is comparable to the time it takes to factor integers based on primes of the same length as in R SA[7]. Recently, it took an estimated 6000 MIPS yews to factor the product of two 429 bit RSA primes[8]. Breaking 512 bit Diffie Hellman would many orders of magnitude more difficult.

Based on these numbers, I think it is safe today that common computer criminals will not be able to launch an effective brute force attack against either of the cryptographic algorithms employed by ETN for at least the next several years. However, organizations with resources comparable to that of a large corporation or small government very possibly could mount a succ .0ss ·1fl 1 brute force cryptographic attack on a recorded ETN session, so one should not transmit information using this protocol that would be worth the cost of attempting such an attack.

Not counting social engineering methods or having super-user privilege on either the client or server machine, it is the author's opinion that the most effective computational method to break ETN is to attack the key generation process. I'll try to explain why, by following the process of attacking the keys backwards from an established session.

Given a recording of a complete session from the network, one can recover the plaintext of the session by knowing (or determining) a session key that is used. Discovering that is computationally expensive at best, especially since some of the most promising known attacks against DES such as Differential Cryptanalysis[9] or Linear Cryptanalysis[10], are unlikely to be feasible given the length and nature of the cyphertext that would be available. Of course, one could easily determine the session key from the data stream and a copy of the source code, if one knows one of the Diffie Hellman private keys used in the agreed key negotiation process. Again, without, this knowledge, it would be computationally intensive at best, to crack the data stream, requiring the calculation of a very large discrete logarithm. However, if one can reproduce the process by which these keys were generated, one doesn't need to do all this math. Again, the algorithm by which the public and private keys are generated is well known, one can make good guesses as to the ease of success by comparing the public value you generate to the one sent over the network in the clear, rather than by trying to decrypt the session stream and see if the results are sensical. These keys are generated by a deterministic process from a data structure acting as a seed consisting of 16 "random" characters with, presumably, 8 bits of randomness in each. By brute force, this is even harder to determine then the DES session keys, but they have to be filled somehow. They are filled using the standard UNIX† `lrand48 (3)` library roll, which produces random enough values over a sequence of 1 G calls, but remember that this routine is seeded by a single number of type `long`, which has only about 4.29 billion possible values.

Unfortunately, that's not the end of the problem. A method is needed to generate this initial seed such that it cannot be guessed by an observer. This is a more difficult task. To make this as secure as possible, there must be some pretty severe constraints on the process by which this number is generated.

1. The distribution of the numbers ought to span the total range of 4.29 billion possible values.
2. The distribution of numbers ought to be fairly even throughout the range of possible values.
3. The number produced should not be guessable by someone observing the session in progress,
4. nor Should the space of values be measurably reduced by the process of observation.
5. The method should be portable to other operating systems and be equally secure on them.
6. The method should not require extra software or privileged instructions.

† As of this writing, the UNIX trademark is owned by X/Open.

7. The method should be fast and require no extra effort on the part of the user.

This is a tall order. We could adopt the method used by 1'C;I'[11] and require the user to type in some amount of text and use the time gaps between key strokes as a source of random information. This would satisfy all but the last criterion, and so it is rejected. We could request a lot of quasi-random information from the system (elapsed uptime, number of free bytes on the "/" disk, a hash of `/etc/motd`, the number of files in `/tmp`, the number of free inodes on the partition with the user's home directory, de.) and hash it by some algorithm to produce a proper seed, but it is unlikely to be portable, may be time consuming and much of the information may be obtainable by another user on one of the machines involved in the connection.

It was decided that time values returned from the system clock (especially the microsecond times) at several points in the code would be used along with the process' PII). These would be combined in the following algorithm (written in C code):

```
seed = ((long) pow(seed[2],pid) + 1000 00*seed [O] + seed[3] - seed[1])%MAXLONG;
```

where `seed [O]`, `seed [1]` and `seed [2]` are microsecond resolution return values and `seed [3]` is a second resolution return value from `gettimeof day` (3). `MAX LONG` is the maximal value of a `long` and `pid` is the return value from `getpid(3)`.

This expression spans the desired range of `long`, is reasonably well distributed, is portable and requires no intervention on the part of the user. The `pid` and `seed [3]` values can be known and guessed at, respectively, by a knowledgeable observer, but that does not provide enough information to be able to narrow the scope of the search by a significant amount.

Unfortunately, the three calls to `gettimeofday` (3) all occur in a relatively small section of code, to keep it modular. Hopefully, one will not be able to correlate the microsecond values well enough to be able to significantly reduce the security of this seed generation process. There may be reason to be concerned on faster machines, as a significant fraction of a second may not have elapsed between each successive call to `get t imeof day` (3). Certainly, a thorough security analysis of this algorithm on several different hardware platforms would be very useful. Fortunately, if it proves to be insecure, the method described above can be replaced with a better one without affecting the protocol at all,

Still, it appears to me that attacking the key generation process is currently the most fruitful line of attack against ETN at this time.

Performance

The initialization is reasonably fast. I clock the time from carriage return to login prompt on an older, heavily loaded Sun SPARC Station to be about 20 to 25 seconds, compared to about 5 seconds for an unencrypted telnet session. The reason that the start up time is so small is that the time consuming part of public key encryption, the generation of primes, does not have to be done in real time. The Diffie Hellman algorithm is secure even with the prime and generator known to the world. Therefore, they are sent over the network by the client in the clear, rather than computed anew each time. The values used are the 512 bit Diffie Hellman prime and generator found in the dhdemo source code from the RSAREF package itself. If one wanted to compute Diffie Hellman keys each time or wished to use RSA to negotiate the session key, one could expect to add about two minutes to the connection time the same machine for 512 bit keys.

The throughput is very good. On the same machine, `cat`-ing a large file (640 KBytes) took 81 seconds over an ETN link to itself, as opposed to 45 seconds in an open Xterm window. I believe the throughput will be more than acceptable for most users.

Standards

The standard available to the Internet community regarding telnet authentication and encryption is RFC 1416. There are RFCs on the specific authentication mechanisms of Kerberos and SPX. There is also a draft RFC on Simple-Strong Authentication (SSA) for telnet, using X.511 as a basis. None of these were particularly applicable to the problems ETN was designed to solve, although SSA comes close.

ETN does *not* conform to the RFC 1416 specification. Here are the reasons why this decision was made: One of the primary uses to which I expect ETN to be put is not as a replacement for telnet, but as a supplementary service to be used only when telnet or rlogin are not secure enough. For example, branch offices of a company connected by the Internet might use rlogin within their local networks and ETN to connect to machines at other offices. Many sites may wish to do router filtering based on the security of the service, allowing ETN through a firewall or gateway while disallowing telnet. Therefore, I do not expect `etnd` to *replace* `telnetd` on any machine. The RFCs provide for fallback to non-encrypted modes when one side is not able or willing to do authentication. I do not want this to ever happen. Users of ETN should *know* that unless they specifically turn it off (and maybe this option should not even be presented,) their session is secure. If it cannot be made secure, it will not be made at all. Furthermore, if one replaces telnet with ETN, there comes the question of how one would mandate an encrypted connection from some hosts while allowing unencrypted telnet sessions from others. With ETN and telnet operating on different port numbers (and providing parallel service) this is easy to do with either router filtering or a package like Wietse Venema's TCP Wrapper[12]. Therefore, it is my belief that ETN should be considered to be a network service that *functions like telnet* but is not telnet, and therefore the RFCs on telnet are not applicable.

There may come a day when it no longer makes sense for there to be a non-RFC 1416 compliant communications mechanism like this. When that day comes, ETN should either be modified to play a new role or abandoned in favor of another system. Alternatively, it may be best that the future of the Internet standards be broadened to include ETN as it stands now. This decision is for others to make.

In my distribution package, I recommend the "standard" use of network port `1005/tcp` for ETN. I also suggest the port `1006/tcp` be reserved for EFTP when it becomes available. These ports have not yet been assigned IANA, the Internet Assigned Numbers Authority, and consequently people who install this package should be willing to change them if it is decided that these port numbers should be changed.

Security Enhancements

There are a number of changes that could be made that may make ETN more secure. One is, easily enough, expanding the protocol to use 1024 bit Diffie Hellman for key exchange, and using triple DES or perhaps IDEA for the session encryption. Another possibility, although it is "security-through-obscuity" is to change the default function used to seed the random number generator at compile time. If it is not possible to discover this from the binaries (or they are installed unreadable) then one cannot effect the sort of attack described in the Vulnerabilities section of this paper. A change of this sort doesn't affect the interoperability of ETN at all. Also, one could have the client and server each be built with the same static Diffie Hellman prime and generator and not have it communicated over the network. It could be hard coded into the binaries, or read from a protected system file. Note that making this change would render the programs incompatible with versions that did not know about this, but it would also speed the startup time of ETN by several seconds. While hiding the prime and generator can buy you some additional security, one should not be too concerned if they are discovered. This is no worse than the default behavior of ETN.

To Do

There are many things yet to be added to ETN. Here are some enhancements that can be expected in the future.

First, ETN's counterpart, EFTP needs to be built. The Diffie Hellman session key negotiation part is modularized in a library and can be added to `ftp` and `ftpd` code very easily. Also, the session encryption process is likely to be much simpler than for telnet, as FTP operates a line at a time rather than character at a time.

Second, I would like to see more available kinds of session key establishment algorithms, perhaps using RSA or LUC as well as Diffie Hellman with different prime lengths. Note that using either RSA or LUC for session key exchange would add considerably to the time ETN would require to initialize, as primes would have to be generated at run time.

Third, the random structure seeding process definitely needs to be enhanced. I hope to solicit more input on making the seed values even less determinable. Perhaps new randomizing functions need to be written. One possible idea is to have an optional daemon running on a system that precomputes several sets of Diffie Hellman or RSA parameters and passes them to the `etn` or `etnd` programs on demand. This daemon could be used by other processes as well. It would be harder to determine the circumstances under which these parameters were generated as well as allowing for the various parts of the seeds to be generated at different times. Also, this would allow for fast implementation of a key exchange using RSA since the computationally expensive processes would have been done long before the exchange takes place.

The event that I think would improve ETN most is for some other organization (or organizations) to do an independent evaluation of the security of the protocol to insure that it has no major weaknesses. It would be unfortunate if some fatal flaw in its function were discovered at a time when it would be difficult for sites who use ETN to upgrade their versions, especially when flaws could probably be fixed fairly painlessly. I would be more than happy to assist any organization who wishes to perform such an evaluation.

Availability

Because of the U.S. export restrictions regarding cryptography, I regret to say that I cannot make this package available to people who are not citizens or permanent residents of the United States or Canada, and I cannot ship this code outside of U.S. and Canadian borders. At this time, I am looking for assistance from some well known anonymous

FTP site in the United States that already has a mechanism in place for distributing code containing restricted encryption who would be willing to house this package and its updates. If your site is interested in assisting in this manner, please contact the author. The redistribution of this package is governed by the copyrights of the individual components, the most limiting of which is the one covering the use of RSAREF. In a nut shell, it says that the ETN package must be distributed with its sourer code and it cannot be used for revenue generating purposes. Please read the RSAREF license agreement for further information.

At this time, ETN has been ported to SunOS 4.1.X and HP-UX 9.0X. Ports to Solaris 2.X and IRIX 5.X are in progress. The author is very interested in working with other individuals on ports to other platforms.

Conclusion

In a time where network sniffers are commonly used by crackers on our networks, our network users deserve an easy to use, high performance protocol to allow them to access remote services without fear of compromise, and our system administrators deserve a way to provide these services to their users that require minimal maintenance. I believe that ETN provides this service and hope that others will find it to 1)(of some benefit.

Acknowledgments

Many people deserve a world of credit for this package. First, the telnet itself is from the BSD **4.3** Net2 distribution. The RSAREF library has been made available by RSA Data Security Inc.. The DES library I used for development comes from Cygnus Support, through the Free Software Foundation. This is the eminent history that this work is based on.

R. Brent Mead and the Institutional Computing and Information Services at the Jet Propulsion Laboratory provided the funding and impetus for this work.

My work started where Seth Robertson of Columbia University left off, porting the BSD Net 2 code to SunOS and linking in the DES libraries. During the early part of the development, a lot of work was done by Jeff Dickson of JPL helping me get the code cleaned up and (10 the preliminary work. Jim Larson of the University of Oregon helped with lots of good advice and one exhaustive hacking session to figure out where to break telnet and telnetd to add the Public Key stuff. Most of all, Peter Scott of JPL deserves more than a world of credit for his tremendous assistance in the latter stages. His expertise saved me many weeks of extra work in getting various parts to execute properly. He should get credit for most all of the pretty code that was added, and, of course, I readily accept all blame for the 11101(brutal hacks.

The work described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

Bibliography

[1] W. Diffie and M.E. Hellman. "New Directions in Cryptography" *IEEE Transactions on Information Theory*, IT-22:644-654, 1976.

[2] CERT Advisory, CA-94:01, available as: f tp: //ftp. cert. erg/pub/ cert_advisories/CA-94:01.ongoing.network monitoring. attacks.

[3] N.M. Haller, "The S/Key One-time Password System" Available via anonymous FTP as:ftp://thumper .bellcore.com/pub/nmh/dots/ISOC .symp.ps.

[4] S.1'. Miller, B.C. Neuman, J.I. Schiller, and J.H. Saltzer. "Kerberos Authentication and Authorization System", *Project Athena Technical Plan*, Section E.2.1, December 21,1987.

[5] D.R. Safford, D.K. Hess, D.L. Schales, "Secure RPC Authentication (SR A) for TEL-NET and FTP", *Proceedings of the Fourth UNIX Security Symposium*, 1993.

[6] M.J. Wiener, "Efficient DES Key Search" *Lecture Notes in Computer Science: Advances in Cryj)tology Crypto'93 Proceedings*. Springer-Verlag, 1994.

[7] 1'. Fahn,Answers to Frequently Asked Questions About Today's Cryptography", Available via anonymous FTP as: ftp://www.rsa.com/pub/faq/faq.asc.

[8] 1). Atkins, "RS - 129", Posting to the Usenet newsgroup sci . crypt on April 26, 1994.

[9] E. Biham and A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 19!33.

[10] M. Matsui, "Linear Cryptanalysis Method for DES Cipher", *Lecture Notes in Computer Science, Advances in Cryptology Eurocrypt '93 Proceedings*, Springer-Verlag, 1994.

[11] P. Zimmerman, "Pretty Good Privacy", Documentation available as part of the PGP Version 2.6.2 software package, See ftp://net-dist.mit.edu/pub/PGP/README for instructions on how to obtain this package. L ast updated October 11, 1994.

[12] W. Venema, "TCP WRAPPER: Network Monitoring, Access Control, and Booby Traps", *Proceedings of the Third UNIX Security Symposium*, 1992.